A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries

François Pellegrini

A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries

François Pellegrini

ENSEIRB, LaBRI and INRIA Futurs
Université Bordeaux I
351, cours de la Libération, 33405 TALENCE, FRANCE
pelegrin@labri.fr

Abstract. Graph partitioning algorithms have yet to be improved, because graph-based local optimization algorithms do not compute smooth and globally-optimal frontiers, while global optimization algorithms are too expensive to be of practical use on large graphs. This paper presents a way to integrate a global optimization, diffusion algorithm in a banded multi-level framework, which dramatically reduces problem size while yielding balanced partitions with smooth boundaries. Since all of these algorithms do parallelize well, high-quality parallel graph partitioners built using these algorithms will have the same quality as state-of-the-art sequential partitioners.

1 Introduction

Graph partitioning is an ubiquitous technique which has applications in many fields of computer science and engineering, such as workload balancing in parallel computing, database storage, VLSI design or bio-informatics. It is mostly used to help solving domain-dependent optimization problems modeled in terms of weighted or unweighted graphs, where finding good solutions amounts to computing, eventually recursively in a divide-and-conquer framework, small vertex or edge cuts that balance evenly the weights of the graph parts.

Many algorithms have been proposed to compute efficient partitions of any graphs, such as graph or evolutionary algorithms, spectral methods, or linear optimization methods. Basically, all of these methods belong to two distinct classes: global methods, which consider all of the graph data, and local optimization heuristics, which try to improve locally a preexisting partition. Global methods often yield better results, but their costs dramatically increases along with problem size, which makes them practically impossible to use for graphs comprising several tens million vertices, which are the graphs now being considered in many scientific engineering problems.

The multi-level approach [5, 6] has been a quite successful attempt to combine both approaches. It consists in repeatedly computing a set of increasingly coarser albeit topologically similar versions of the graph to partition, by finding

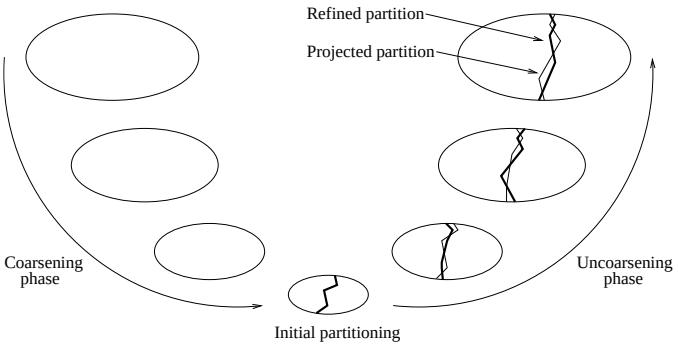


Fig. 1. Multi-level framework for computing a bipartition of a graph.

matchings which collapse vertices and edges, until the coarsest graph obtained is no larger than a few hundreds of vertices, then computing a separator on this coarsest graph, and projecting back this separator, from coarser to finer graphs, up to the original graph. Most often, a local optimization algorithm, such as Kernighan-Lin [7] or Fiduccia-Mattheyses [4] (FM), is used in the uncoarsening phase to refine the partition that is projected back at every level, such that the granularity of the solution is the one of the original graph and not the one of the coarsest graph, as illustrated in Figure 1. This approach improves quality over plain graph algorithms, and speed over plain global optimization algorithms, by taking the best of both worlds. Global optimization algorithms can be used on small graphs to give the general direction of the partition to set, and inexpensive local optimization algorithms can be used at low cost on finer graphs with tens of million vertices.

However, the quality of partitions produced by this approach is not as good as the one that would be yielded by plain global optimization algorithms. Coarsening artifacts, as well as the meshing topology of the original graphs, trap local optimization algorithms in local optima of their cost functions, such that frontiers are often made of non-optimal sets of segments, as illustrated in Figure 5.a.

This paper describes an efficient way to integrate diffusion schemes into a multi-level framework, so as to compute partitions with small and smooth frontiers in a time equivalent in magnitude to the one of state-of-the-art local optimization algorithms. It is organized as follows. After presenting related works in Section 2, we introduce in Section 3 our multi-level banded diffusion scheme, and show some partitioning and mapping results, obtained with Scotch 5.0, in Section 4. Then comes the conclusion.

2 Related works

Many authors had already noticed that partitions yielded by local optimization algorithms were not optimal. One of the most vocal communities was the one

of the users of iterative linear system solving methods [12], which experienced that such partitions were not fitted for their purpose, as subdomains with longer frontiers or irregular shapes resulted in a larger number of iterations to achieve convergence. To measure the quality of each of the parts, several authors defined a metric called *aspect ratio*, which can be thought in 2D as a measure of the perimeter of a part with respect to the square root of its area. The more compact a part is, the smaller its aspect ratio value is, as ideal parts are of circular shape in the Euclidean space.

In [3], Diekmann *et al.* evidenced such a behavior, and proposed both a measure of the aspect ratio of the parts, as well as a set of heuristics to create and refine the partitions, with the objective of decreasing their aspect ratio. Among these algorithms is a "bubble-growing" algorithm. This algorithm is based on the observation that sets of soap bubbles self-organize so as to minimize the surface of their interfaces, which is indeed what is expected from a partitioning algorithm. Consequently, the authors' idea was to grow, from as many seed vertices as the desired number of parts, a collection of expanding bubbles, by performing breadth-first traversals rooted at these seed vertices. Once every graph vertex has been assigned to some part, each part computes its center based on the graph distance metric. These center vertices are taken as new seeds and the expansion process is started again, until it converges, that is, until centers of subdomains no longer move. An important drawback of this method is that it does not guarantee that all parts will hold the same number of vertices, which requires to call other heuristics in turn to perform load balancing. Also, all of the graph vertices must be visited many times, which makes this algorithm quite expensive, all the more it is combined with costly algorithms such as simulated annealing, and the computation of the aspect ratio requires some knowledge on the geometry of the graphs, which is not always available.

In [8], Meyerhenke and Schamberger further explore the bubble model, and devise a way to grow the bubbles by solving, possibly in parallel, systems of linear equations, instead of iteratively computing bubble centers. This method yields partitions of high quality too, but is very slow, even in parallel [9], and the load balancing problem is also not addressed, which requires to resort to a greedy load balancing algorithm afterwards.

In [13], Wan *et al.* explore a diffusive model, called the *influence model*, where vertices impact their neighbors by diffusing them information on their current state. This model also does not handle load balancing properly.

3 Multi-level banded diffusion scheme

In spite of their better quality, all of the above diffusion schemes have two drawbacks: first, they do not naturally balance loads between parts and second, they are expensive as they involve all of the graph vertices. The method that we propose in this paper addresses both of these problems.

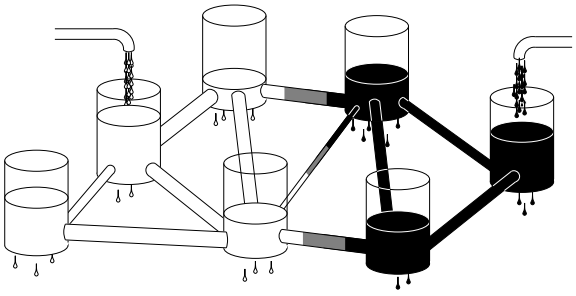


Fig. 2. Sketch of our diffusion model.

3.1 The jug of the Danaides

The diffusion scheme that we propose can apply to an arbitrary number of parts, but for the sake of clarity we will describe it in the context of graph bipartitioning, that is, with two parts only. We model the graph to bipartition in the following way, depicted in Figure 2. Nodes are represented as barrels of infinite capacity, which leak such that one unit of liquid at most drips per unit of time. When graph vertices are weighted, always with integer weights, the maximum quantity of liquid to be lost per unit of time is equal to the weight of the vertex. Graph edges are modeled by pipes of section equal to their weight. In both parts, a source vertex is chosen, to which a source pipe is connected, which flows in $\frac{|V|}{2}$ units of liquid per unit of time. Two sorts of liquids are in fact injected in the system: scotch in the first pipe, and anti-scotch in the second pipe, such that when some quantity of scotch mixes with the same quantity of anti-scotch, both vanish. To ease the writing of the algorithm in the bipartitioning case, scotch is represented by positive quantities and anti-scotch is represented by negative ones, so that mutual destruction naturally takes place when adding any two quantities of opposite signs.

The diffusion algorithm performs as outlined in Figure 3. For each time step, and for each vertex, the amount of liquid (whether scotch or anti-scotch) which remains after some has leaked is spread across the connecting pipes towards the neighboring barrels, according to their relative sections. This process could be iterated until convergence, but in fact it is only performed for a number of steps sufficient to achieve sign stability. Indeed, we are not interested in complete convergence, but in the stability of the signs of all content quantities borne by graph vertices, which indicate whether scotch or anti-scotch dominates in the barrels, that is, if some vertex belongs to part 0 or 1.

Since $|V|$ units of both liquids are injected on the whole per unit of time, and since all of the barrels can leak the same overall amount in the same time, the system is bound to converge, all the more that liquid can disappear by collision of scotch and anti-scotch. As in the bubble schemes, what is expected is that a smooth front will be created between the two parts. The purpose of the algorithm is more to have a global smoothing of the frontier than a strict minimization of

```
while (number of passes to do) {
  reset contents of new array to 0;
  old[s0] ← old[s0] − |V|/2;          /* Refill source barrels */
  old[s1] ← old[s1] + |V|/2;
  for (all vertices v in graph) {
    c ← old[v];                       /* Get contents of barrel            */
    if (|c| > weight[v]) {            /* If not all contents have leaked   */
      c ← c − weight[v] ∗ sign(c);    /* Compute what will remain          */
      σ ← Σ_{e=(v,v')} weight[e];     /* Sum weights of all adjacent edges */
      for (all edges e = (v, v')) {   /* For all edges adjacent to v       */
        f ← c ∗ weight[e]/σ;          /* Fraction to be spread to v'       */
        new[v'] ← new[v'] + f;        /* Accumulate spreaded contributions */
      }
    }
  }
  swap old and new arrays;
}
```

Fig. 3. Sketch of the jug-of-the-Danaides diffusion algorithm. Scotch, represented as positive quantities, flows from the source of part 1, while anti-scotch, represented as negative quantities, flows from the source of part 0. For each step, the current and new contents of every vertex are stored in arrays `old` and `new`, respectively.

the cut. In fact, unlike all of the algorithms presented in the previous section, our method privileges load balancing over cut minimization. For this latter criterion, we rely on an additional feature of our scheme, as explained below.

3.2 Band graphs in a multi-level scheme

Our diffusion algorithm, as such, presents two weaknesses: nothing is said about the selection of the seed vertices, and performing such iterations over all of the graphs vertices is very expensive compared to local optimization algorithms which only consider vertices in the immediate vicinity of the frontiers.

To address these two problems concurrently, we use a method we have developed in [1], illustrated in Figure 4. It consists in using a multi-level scheme in which refinement algorithms are not applied to the full graphs but to band graphs that contain vertices that are at most at some small distance, typically 3, from the projected separator. In these band graphs, two additional "anchor" vertices represent all of the removed vertices of each part, and are connected to the last band layers of vertices of each of the parts. The vertex weight of the anchor vertices is equal to the sum of the vertex weights of all of the vertices they replace, to preserve the balance of the two band parts.

The underlying reasoning of this pre-constrained banding scheme is that since every refinement is classically performed by means of a local algorithm, which perturbs only in a limited way the position of the projected separator, local refinement algorithms need only to be passed a subgraph that contains the vertices that are very close to the projected separator. We have experimented that,

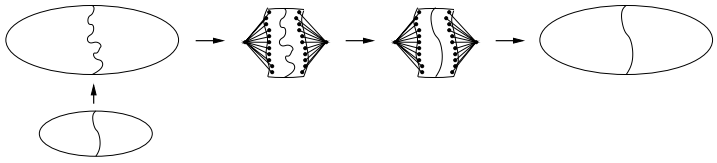


Fig. 4. Multi-level banded refinement scheme. A band graph of small width is created around the projected finer separator, with anchor vertices representing all of the removed vertices in each part. After some optimization algorithm (whether local or global) is applied, the refined band separator is projected back to the full graph, and the uncoarsening process goes on.

when performing Fiduccia-Mattheyses refinement on band graphs that contain only vertices that are at distance at most 3 from the projected separators, the quality of the finest separator not only remains constant, but even significantly improves in most cases. Our interpretation is that this pre-constrained banding prevents local optimization algorithms from exploring and being trapped in local optima that would be too far from the global optimum sketched at the coarsest level of the multi-level process.

Such a banded scheme is ideal for using our diffusion scheme, as anchor vertices represent a natural choice to be taken as seed vertices. Indeed, the most important problem for bubble-growing algorithms is the determination of the seed vertices from which bubbles are grown, which requires expensive processes involving all of the graph vertices [3, 8]. Since anchor vertices are connected to all of the vertices of the last layers, the diffused liquids flow as a front as if they originated from the farthest vertices from the frontier, which is indeed what would happen if they flowed from the center of a bubble having the frontier as its perimeter.

3.3 Parallelization

Our diffusion algorithm has the additional interest of being highly scalable. If we assume that full graphs, as well as band graphs, are distributed across processors such that every processor holds a fraction of the graph vertices along with their adjacency lists, like what is done for instance in PT-Scotch [2], the parallel version of Scotch, the parallel version of the algorithm is straightforward. Every processor performs its local update and computes the contributions it has to spread to distant neighbors, after which these contributions are sent to their destination processors in order to be aggregated. In order to cover communication by computations, vertices that have distant neighbors can be processed first, then communications are started, and vertices with purely local adjacency lists can be processed in the mean time, before received contributions are aggregated.

Table 1. Description of the test graphs that we use, which all relate to 3D problems, except *thread*. $|V|$ and $|E|$ are the vertex and edge cardinalities, in thousands.

Graph	Size ($\times 10^3$)		Average degree
	$\|V\|$	$\|E\|$	
altr4	26	163	12.50
audikw1	944	38354	81.28
auto	449	3315	14.77
bmw32	227	5531	48.65
body	45	164	7.26
bracket	63	367	11.71
conesphere1m	1055	8023	15.21
ocean	143	410	5.71
oilpan	74	1762	47.77
pwt	37	145	7.93
thread	30	2220	149.32

4 Experimental results

The diffusion algorithm discussed above has been implemented, as a sequential graph bipartitioning method, in version 5.0 the Scotch [10] graph partitioning and static mapping software. Its k-way implementation is not yet available, because it requires more coding, including a k-way band extraction algorithm which does not exist to date. All of the necessary floating-point arithmetic has been implemented in single precision.

The tests were run on a Lenovo ThinkPad T60 laptop, with an Intel dual-core T2400 processor running at 1.8 MHz and 1 Gb of memory. As we ran sequential tests only, the dual-core feature of the processor is not relevant. The test graphs we have used in our experiments are listed in Table 1. These graphs were partitioned into 2 to 128 parts, and the three quality metrics that we consider are the number of cut edges, called **Cut**, a load imbalance ratio equal to the size of the largest part divided by the average size, called **MaCut**, and the maximum diameter of the parts, referred to as **MDi**, which is an indirect metric of the shape of the partition, and is usable even in the case of graphs of unknown or nonexistent geometry. This latter metric is insufficient, as it does not really capture the smoothness of the interfaces, since irregularly shaped parts can still have small diameters; the best proof would have been to run an iterative solver and measure convergence rates basing on the numbers of iterations. This work is in progress.

Three diffusion heuristics were compared against the classical strategy implemented in Scotch 4.0, referred to as RMF in the following, which performs recursive bipartitioning with bipartitions computed in a multi-level way, using FM refinement.

The first method, RMBD, uses the same recursive bipartitioning and multi-level strategy, but banded diffusion is performed during the multi-level refinement steps. The results achieved with this method validate our approach: the obtained partitions have very smooth boundaries (see Figure 5.b), and are adequately balanced if the number of diffusion iterations is sufficiently high, as shown in Table 2.

Table 2. Evolution of the cut size (Δ**Cut**), of the load imbalance ratio (Δ**MaCut**) and of the maximum diameter of the parts (Δ**MDi**) produced by various partitioning heuristics with respect to the RMF strategy, averaged over all test graphs and numbers of parts. Figures below partitioning strategy names indicate the number of diffusion steps performed.

Method	RMBD				RMBDF		RMBaDF
	500	200	100	40	500	40	40
Δ**Cut** (%)	+19.51	+20.01	+18.15	+21.49	+2.26	+3.10	-3.17
Δ**MaCut** (%)	+0.58	+1.12	+1.80	+9.76	-0.95	-0.29	-0.21
Δ**MDi** (%)	+3.86	+1.92	+4.69	+5.43	+2.26	+3.10	-3.24
ΔTime ($\times$)	21.31	9.33	5.33	2.93	21.47	2.99	3.07

When performing 100 diffusion steps, the average **MaCut** value for RMBD is 1.046, only 1.80 % higher than the one of RMF. However, the maximum diameter **Mdi** is not significantly reduced, and is even increased on average by 4.69% with respect to RMF. This method is also 5.33 times slower than RMF and increases the cut by about 20%, which makes it of little practical use.

We have therefore experimented a second method, RMBDF, where the classical FM algorithm is applied to the band graph after the diffusion algorithm. The idea of this strategy is to benefit from the global optimization capabilities brought by the diffusion algorithm, while locally optimizing the frontier afterward. Even when performing 40 diffusion steps only, the smoothness of the boundaries is preserved and parts are more balanced, while the cut is only increased by 3.10% with respect to RMF. This strategy is also only three times slower than RMF, which is extremely fast for a diffusion-based algorithm.

In order to favor the minimization of diameters, we have modified our diffusion method so as to double at each step the amount of liquid borne by every vertex, in an "avalanche"-like process. This method is referred to as "aD". It is no longer bound to converge, and indeed causes overflows for large numbers of diffusion steps, but gives good results for small numbers of iterations. As a matter of facts, we can see in Table 2 that the RMBaDF method is the most efficient one on average, and yields better results than the classical RMF method while still providing smooth boundaries, as evidenced in Figure 5.c.

For the sake of comparison, we compare in Table 3 some of our results against the ones obtained with K-MeTiS. K-MeTiS uses direct k-way partitioning instead of recursive bipartitioning, which usually makes it more efficient when the number of parts increases, and also much faster (from 10 to 20 times). As analyzed in [11], the performance of recursive bipartitioning methods tends to decrease when the number of parts increases, which should limit the efficiency of RMBDF methods for large numbers of parts. A full k-way diffusion algorithm is therefore required.

Table 3. Comparison of the results, in terms of cut size (**Cut**) and maximum diameter of the parts (**MDi**), between three heuristics: multi-level with FM refinement (RMF, as implemented in Scotch 4.0), multi-level with banded diffusion and FM refinements (RMBaDF), and K-MeTiS.

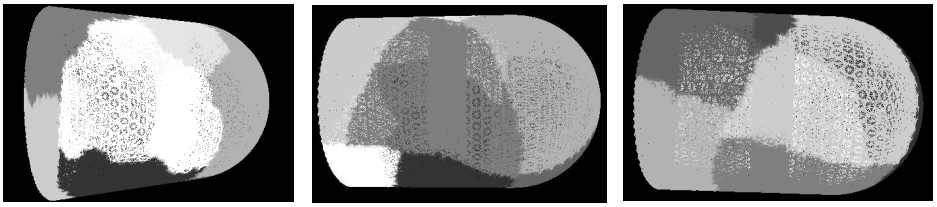
Test case		Number of parts						
		2	4	8	16	32	64	128
altr4								
RMF	**Cut**	1688	**3197**	**4978**	**7788**	**11905**	17656	24478
	MDi	50	52	**40**	33	**25**	21	**14**
RMBaD(40)F	**Cut**	**1621**	3203	5017	7776	11980	17669	24831
	MDi	**48**	46	41	**30**	**25**	**18**	**14**
KMeTiS	**Cut**	1670	3233	4981	8115	12147	**17355**	**24058**
	MDi	**48**	**45**	41	34	26	22	**14**
bmw32								
RMF	**Cut**	17271	**54424**	84222	**120828**	**181844**	**267427**	**394418**
	MDi	93	116	130	106	74	120	68
RMBaD(40)F	**Cut**	16032	54446	**83422**	124945	183454	275594	411154
	MDi	91	130	**96**	**84**	**68**	63	**56**
KMeTiS	**Cut**	**15529**	55506	92658	125686	193169	286111	420965
	MDi	**87**	**108**	99	87	70	**61**	68

5 Conclusion and future work

In this paper, we have presented a diffusion algorithm which, used in a multi-level banded framework, results in smoother partition frontiers and more compact parts. Used in our banded context, this algorithm is fast enough to be used on very large graphs, as it is only about three times slower than classical local optimization schemes. The 2-way sequential version has been integrated in version 5.0 of Scotch.

This algorithm is also easily parallelizable and highly scalable, which makes it a very good candidate for the realization of a fast and efficient parallel graph partitioner, taking advantage of the parallel multi-level and band graph extraction routines already developed in PT-Scotch in the context of sparse matrix reordering.

Even more than classical FM-like algorithms, this algorithm is constrained by the greedy nature of the recursive bipartitioning scheme, which prevents the global improvement of frontiers computed at previous stages. A full k-way version of the algorithm is therefore under development, which extends the 2-way model by considering k different liquids having the same mutual annihilation properties, such that when p different liquids are mixed in the same barrel, only the most abundant one remains. This behavior is equivalent to the one of our algorithm in the 2-way case. Using a native k-way scheme should also significantly reduce running times compared to recursive bipartitioning. A parallel version is also being developed.



a. RMF. **b.** RMBD. **c.** RMBaDF.

Fig. 5. Partition of graph **altr4** into 8 parts using three different strategies. The segmented frontiers produced by FM-like algorithms are clearly evidenced in Figure **a**. RMBD produces the smoothest boundaries, as shown in Figure **b**. RMBaDF takes the best of both worlds, in Figure **c**.

References

1. C. Chevalier and F. Pellegrini. Improvement of the efficiency of genetic algorithms for scalable parallel graph partitioning in a multi-level framework. In *Proc. Europar*, pages 243–252, 2006. `http://www.labri.fr/~pelegrin/papers/scotch_efficientga.pdf`.
2. C. Chevalier and F. Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. *Submitted to Parallel Computing*, dec 2006. `http://www.labri.fr/~pelegrin/papers/scotch_parallelordering_parcomp.pdf`.
3. R. Diekmann, R. Preis, F. Schlimbach, and C. Walshaw. Aspect ratio for mesh partitioning. In *Proc. Europar'98, LNCS 1470*, pages 347–351, 1998.
4. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. 19th Design Automat. Conf.*, pages 175–181. IEEE, 1982.
5. B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of Supercomputing*, 1995.
6. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. on Scientific Computing*, 20(1):359–392, 1998.
7. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitionning graphs. *BELL System Technical Journal*, 49:291–307, feb 1970.
8. H. Meyerhenke and S. Schamberger. Balancing parallel adaptive FEM computations by solving systems of linear equations. In *Proc. Europar*, pages 209–219, 2005.
9. H. Meyerhenke and S. Schamberger. A parallel shape optimizing load balancer. In *Proc. Europar'2006, LNCS 4128*, pages 232–242, 2006.
10. Scotch: Static mapping, graph partitioning, and sparse matrix block ordering package. `http://www.labri.fr/~pelegrin/scotch/`.
11. H. D. Simon and S.-H. Teng. How good is recursive bipartition. *SIAM J. Scientific Computing*, 18(5):1436–1445, sep 1997.
12. R. Vanderstraeten, R. Keunings, and C. Farhat. Beyond conventional mesh partitioning algorithms. In *SIAM Conf. on Par. Proc.*, pages 611–614, 1995.
13. Y. Wan, S. Roy, A. Saberi, and B. Lesieutre. A stochastic automaton-based algorithm for flexible and distributed network partitioning. In *Proc. Swarm Intelligence Symposium*, pages 273–280. IEEE, 2005.